ISLAMIC UNIVERSITY OF GAZA
FACULTY OF ENGINEERING
ELECTRICAL AND COMPUTER DEPARTMENT



SIGNALS AND SYSTEMS LABORATORY

Prepared by:

AHMAD AL KHATEB
FADWA El SHAWAF

Introduction

Objective of Signals and Systems Lab

There are experiments are selected based upon the course material, most of the Signals and Systems course is oriented toward signal analysis.

The objective of the experimental laboratory is to provide students the experience with methods of measurement and data analysis and encourage the students to use MATLAB to write their programs.

Each work is expected to be turned in one week from the time that the experiment was assigned.

Each week the student must submit the lab report for the previous one and late reports will NOT be accepted at any time under any case. Don't Forget "An ill workman blames his tools"

Each week Quizzes are supposed according to your alacrity.

You must know that allude and amenable students are preferable anyway.

Grading Policy

Laboratory grading will be divided into four categories, Reports, quizzes, and Exams.

Reports	40 %
Quizzes	15 %
Midterm Exam	20 %
Final Exam	25 %
Total	100 %

Lab Schedule

Weeks	Experiments	Page
1	Introduction & Syllabi	
2	Experiment 1: (Introduction to MATLAB)	5
3	Experiment 2: (Properties of Discrete-Time systems)	8
4	Experiment 3: (Linear Time-Invariant Systems)	10
5	Experiment 4: (Linear Time-Invariant Systems)	13
6	Experiment 5: (Fourier Series)	16
7	Experiment 6: (Fourier Transform)	19
8	Midterm Exam	
9	Experiment 7: (Laplace Transform)	21
10	Experiment 8: (Frequency Response and Filters)	23
11	Experiment 8: (Frequency Response and Filters)	23
12	Experiment 9: (Frequency Response (Simulink))	26
13	Experiment 10: (Fourier Series (Simulink))	32

LABORATORY SAFETY

Experiments in this laboratory will be conducted in strict accordance with following list of regulations, procedures, and comments in order to promote a professional and safe approach to the laboratory experience. Additionally, laboratory safety rules apply during all experiments. If you are not sure of the operation of equipment or laboratory procedure, particularly those which might compromise personal safety and the safety of your laboratory partners, do not hesitate to ask your laboratory instructor for assistance**. The following rules must be strictly adhered to** during the course of your laboratory experiment:

1. No smoking, No food, and No drinks permitted inside the Laboratory.
2. Wear proper clothing and insulated footwear to the laboratory.
3. Do not use wet hands or stand on a wet floor while making electrical connections.
4. Do not place personal belongings (books, coat, etc.) on laboratory equipment.
5. Keep your work area clean and organized.
6. Use only those equipment required for a particular experiment specified.
7. Do not use damaged or poorly insulated wires or equipment
8. Properly ground all equipment.
9. Thoroughly check all connections before applying power.
10. Turn power off when making changes to your experiment.
11. Discharge capacitors by shorting with resistor.
12. Do not energize equipment until give permission.
13. Do not touch 120 Volt electrical outlets or the terminals of any energized electrical connection.
14. Report any accident to your instructor immediately.
15. Work deliberately and carefully.
16. In the event of a power failure, turn-off the power switches to all equipment immediately and await further instructions.
17. After you are done with your experiment, turn all main switches off.
18. Failure to follow safety instructions can cause serious bodily injury or death.

I hereby certify that I have read the above safety requirements and agree to follow all safety procedures in this laboratory.

______________________	______________________	______________
Print Name	Signature	Date

Experiment # 1

Introduction to MATLAB

1- Objective

The objective of this lab is to introduce you to MATLAB software and allows you to become familiar with some of the main built in functions that you will use in the following programs.

2 Basic Theoretical Principles:

2.1 Characteristics of the MATLAB Environment

The MATLAB was originally developed to be the *Matrix Laboratory*. Today's MATLAB with capabilities far beyond the original MATLAB, is an interactive system and programming language for general scientific computation. Its basic elements is a matrix. MATLAB commands are similar to the expression of engineering steps in mathematics. This make it easier and quicker to write a program in MATLAB.

2.2 MATLAB Windows:

MATLAB has three display windows:

1- A command window is used to enter commands, data, and to print results.
2- A graphic window is used to display plots and graphics.
3- An editing window is used to write codes (M-Files).

When you first start MATLAB, the command window will be the active window. You will get a MATLAB **prompt** (>>), you can enter any MATLAB command at this prompt. To exit this windows, you can select *exit MATLAB* from the file menu, or type **exit** or **quit** at the MATLAB prompt.

2.3 Writing Codes Using MATLAB

The first thing to remember is that MATLAB is *case sensitive*.

To write a code in MATLAB, go to the file menu and select **New**, then **M-file.** Once the M-file selected, you will get an editor window. You are now ready to write your code. To write a code, you need to be familiar with the MATLAB commands. The more you practice, the more you will be able to write those codes (this is true with any programming language). Once you have written your code, you need to save it with the extension (**.m**), as an example, **Exple_1.m** (*Note*: It is worth mentioning that sometimes, the default in MATLAB is set to the extension **.txt**, so make sure that you have saved your file with the extension **.m**).

Once, the code has been written and saved, you are ready to run it. To do that, you need to go from the editing window to the command window. At the MATLAB prompt (>>), you need to type the file name.

Note: To run a program, make sure that the file you need to run is in the same directory as the directory used in the command window (working directory). To check the current working directory type **pwd** (**P**resent **W**orking **D**irectory) at the MATLAB prompt. After typing the command **pwd**, you can get an output such as **C:\MATLAB\BIN** this shows that you are can run any program in the subdirectory **BIN**. If you try to run a program from another directory while you command window has another directory location (different from the one containing you file) you will get an error message. To change directory you type **cd ..** to go one level down or **cd** and then the directory name such as **cd a:** to go the A drive.

2.4 MATLAB Basic Commands

These are the basic commands that will be used in the following programs. You need to make yourself familiar with these commands. These commands can be added to any code and can be typed at the MATLAB prompt (>>).

The **clc** command clears the command window.

The **clear** command does not affect the command window, but it does remove all variable from memory. It is usually a good idea to start your code with the **clear** command.

The **plot** command allows you to plot graphs from data stored in vectors. There are many options that goes with the **plot** command. You will be familiar with these options at a later stage.

The **stem** command generates a point plot with lines connecting the points to the x-axis.

The **ones** command and the **zeros** commands are similar, the only difference is that the **ones** command generate a vector containing 1, and the **zeros** command generate a vector containing 0. As an example **ones(1,5)** will generate a row vector of length 5 with all the elements of the vector being 1 and **zeros(5,1)** will generate a column vector with all the elements of the vector being 0. You can generate an n x m matrix with all the elements being zeros or ones by typing zeros(n,m) respectively ones(n,m).

3- Experiment Procedure:

Given the following code written in MATLAB:

% This is code to plot the function $y = x^2$

```
clear
x=[-5:1:5];
y=x.^2;
plot(y);
grid;
title('Program 1. Plot of the function y=x.^2')
xlabel('x');
ylable('y');
```

Write this code and run it using MATLAB.

1. Need to provide the plot for the code above.
2. Next, in the above code, replace *plot(y)* by *plot(x,y).* What is the difference between the two plots? Comment on your results.
3. Replace *plot(y)* by *plot(y,x).* What is the difference between the original and the current plots? Comment on your results.
4. In the above code, omit the semi column from the following expression *x=[-5:1:5];* i.e. *x=[-5:1:5]*. Rerun your program. What difference does this change make in running your program?
5. Add the percentage sign in front of the grid command, i.e., *% grid* and run your program. Does this change affect the output of your program? Comment on that.
6. Replace the *plot(y);* by *stem(y)* and rerun the program. What is the difference between the two plots?

Next, you need to write an expression using the **ones** and **zeros** command to generate a vector of length 200 containing the following elements (Notice: This is very important for your class work, it allows you write expression for the unit step and the delta function):

$$y(n) = \begin{cases} 0 & \text{for } 0 \le n \le 50 \\ 2 & \text{for } 51 \le \text{n} \le 150 \\ -0.5 & \text{for } 151 \le n \le 200 \end{cases}$$

Next right a code using MATLAB to plot y(n) using the stem command. Your plot should contains title, x-label, y-label, grid, and you need to write the following phrase on your plot: "*Program 1. Part 2*". Notice to do this, you need to use the **gtext** command.

Notes:
There are helpful commands you can use using MATLAB:
help: you can type this at the command windows. This gives a directory of all toolbox and commands used in MATLAB. You can type **help** followed by a command to check what such command does. As an example, type *help clear*.
Lookfor: this commands allows you to look at a particular subject, not necessarily a command. As an example, *lookfor graphics*.
You can abort a program by pressing the Ctrl C.
The student version of MATLAB (Version 4) is very similar to the professional Version 4 (commend wise) but it has the following limitations:

1. Each vector is limited to 8192 elements.
2. Each matrix is limited to a total of 8192 elements with either the number of rows or columns limited to 32.
3. Programs cannot be dynamically link C or Fortran subroutines.

Experiment # 2

Properties of Discrete-Time systems

Implementing First-Order difference Equation

Discrete-time systems are often characterized in terms of a number of properties such as linearity, time invariant, stability, causality, and invertability.
Discrete-time systems are often implemented with linear constant-coefficient difference equations. Two very simple difference equations are the first-order moving average

$$y[n] = x[n] + bx[n-1],$$

and the first order autoregression

$$y[n] = ay[n-1] + x[n],$$

In this program you are asked to write a program (function) which implements the first-order autoregression equation. You will then be asked to test and analyze your function on some example systems.

a-
Write a function *y=diffeqn(a,x,yn1)* which computes the output y[n] of the causal system determined by the equation:

$$y[n] = a*y[n-1] + x[n] \quad (1)$$

The input vector x contains x[n] for 0 <= n <= N-1 and yn1 supplies the value y[-1]. The output vector y contains y[n] for 0 <= n <= N-1. The first line of M-file should read:

function y = diffeqn(a,x,yn1)

Hint: Note that y[-1] is necessary for computing y[0], which is the first step of the autoregression. Use a for loop in your M-file to compute y[n] for successively larger values of n, starting with n = 0.

b-
Assume that a =1, y[-1] = 0, and that we are only interested in the output over the interval 0 <= n <= 30. Use your function to compute the response due to $x1[n] = \delta[n]$ and x2[n]=u[n], the unit impulse response and the unit step respectively. Plot each response using *stem*.

c-
Assume again that a=1, but y[-1] = -1. Use your function to compute y[n] over 0<=n<=30 when the input are x1[n] = u[n] and x2[n] = 2.u[n]. Define the output produced by the two signals to be y1[n] and y2[n], respectively. Use *stem* to display both outputs. Use stem to plot (2*y1[n]-y2[n]). Given that equation (1) is linear difference equation, why isn't this difference equation identically zero?

d-

The causal system describe in equation (1) are BIBO (bounded-input bounded-output) stable whenever $|a| < 1$. A property of this stable systems is that the effect of the initial condition becomes insignificant for sufficiently large n. Assume $a = ½$, and that x contains $x[n]=u[n]$ for $0 <= n <= 30$. Assuming both $y[-1] = 0$ and $y[-1] = ½$, compute the two output signals $y[n]$ for $0 <= n <= 30$. Use *stem* to display both responses. How do they differ?

Experiment # 3

Linear Time-Invariant Systems

Output of LTI systems

This program explain how to use MATLAB to compute the output of LTI systems using the function *conv*.

The MATALB function *conv* computes the convolution sum

$$y[n] = \sum_{m=-\infty}^{\infty} h[m]x[n-m]$$

assuming that x[n] and h[n] are finite-length sequences. If x[n] is nonzero only on the interval nx <= n <= nx + Nx - 1 and h[n] is nonzero only on the interval
nh < = n < = nh + Nh - 1, then y[n] can be nonzero only on the interval

(nx + nh) <= n <= (nx + nh) + Nx + Nh - 2,   (1)

meaning that *conv* needs only compute y[n] for the Nx + Nh - 1 samples on this interval. If x is an Nx-dimensional vector containing x[n] on the interval nx <= n<=nx +Nx-1 and h is an Nh-dimensional vector containing h[n] on the interval nh <= n <= nh + Nh - 1, then y=*conv*(h,x) returns in y the Nx + Nh - 1 samples of y[n] on the interval in Eq. (1). However, *conv* does not return the indices of the samples of y[n] stored in y, which makes sense because the intervals of x and h are not input to *conv*.

a-

Consider the finite length-length signal

$$x[n] = \begin{cases} 1, & 0 \le n \le 5, \\ 0, & \text{otherwise} \end{cases} \qquad (2)$$

Analytically determine y[n] = x[n]*x[n].

b-

Compute the nonzero samples of y[n] = x[n]*x[n] using *conv*, and store these samples in the vector y. Your first step should be to define the vector x to contain the samples of x[n] on the interval 0 <= n <= 5. Also construct an index vector ny, where ny(j) contains the index of the sample of y[n] stored in the j-th element of y, i.e., y(j) = y[ny(j)]. For example, ny(1) should contain $n_x + n_x$, where nx is the first nonzero index of x[n]. Plot your result using *stem(ny,y)*, and make sure that your plot agrees with the signal determined in Part (a). As a check, your plot should also agree with figure 3.1.

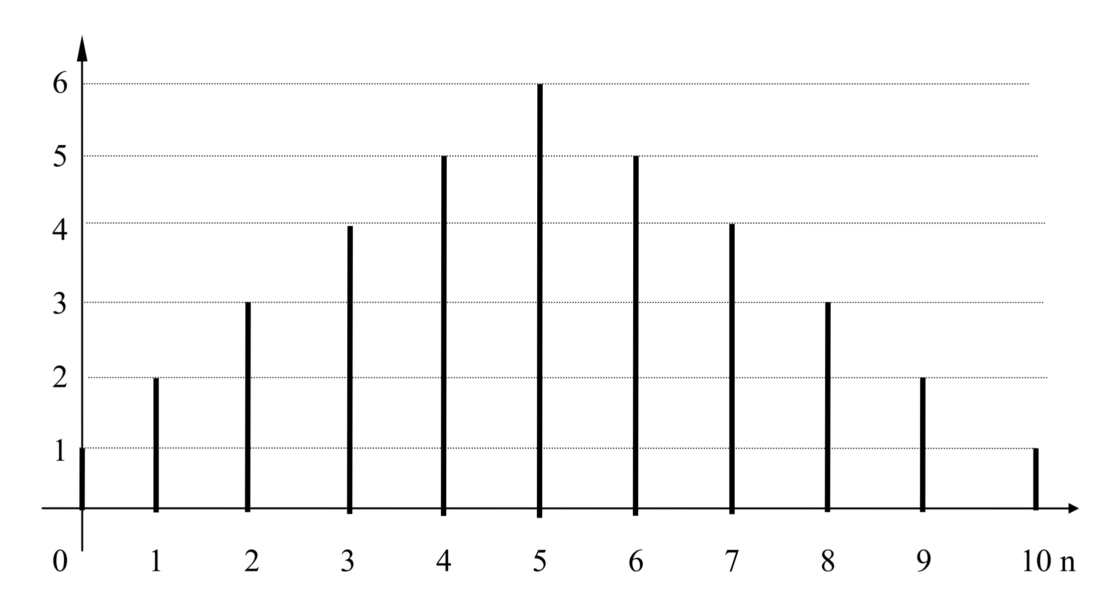


Fig: 3.1 Plot of the signal y[n]=x[n]*h[n]

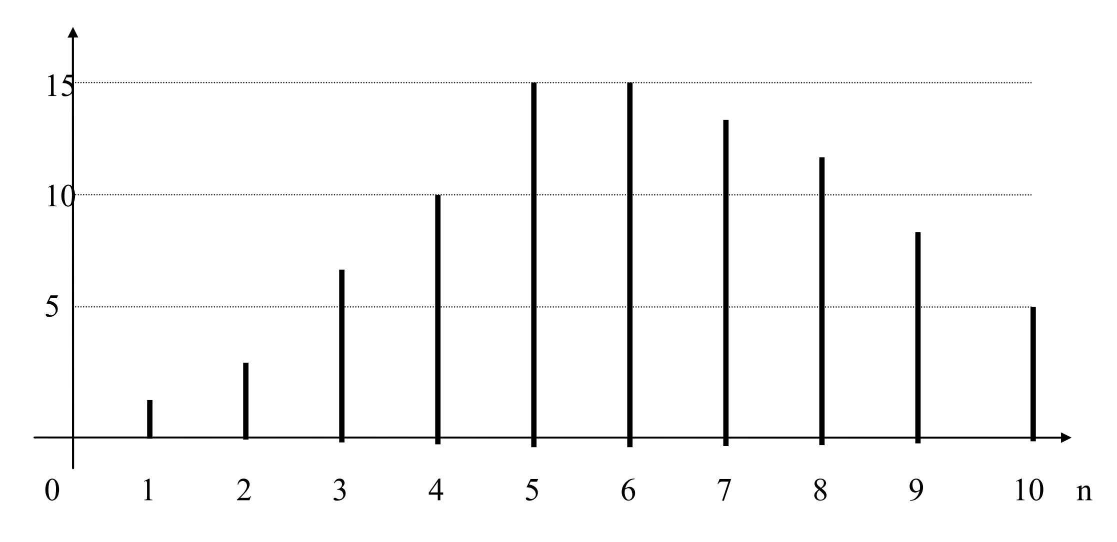


Fig: 3.2 Plot of the signal y[n]=x[n]*h[n]

c-

Consider the finite-length signal

$$h[n] = \begin{cases} n, & 0 \le n \le 5, \\ 0, & \text{otherwise} \end{cases} \qquad (3)$$

Analytically compute y[n] = x[n] * h[n]. Next compute y using *conv*, where your first step should be to define the vector h to contain h[n] on the interval 0 <= n <= 5. Again construct a vector ny which contains the interval of n for which y contains y[n]. Plot your results using *stem*(ny,y). As a check, your plot should agree with figure 3.2 and your analytical derivation. For the example computed just considered implementing y[n] = x[n] * h[n] using *conv* the signal h[n] can be viewed as the impulse response of a linear time-invariant system for which x[n] is the system input and y[n] is the system output. Because h[n] is nonzero for n<0, this system is causal. However, *conv* can also be used to implement LTI systems which are noncausal. You must still be careful to keep track of the indices of x[n], h[n], and y[n]. For example, consider the system with impulse response h[n+5], where h[n] is defined in equation (3).

d-

How does y2[n] = x[n] * h[n + 5] compare to the signal y[n] derived in Part ©?

e-

Use *conv* to compute the nonzero samples of y2[n], and store these samples in the vector y2. If done correctly, this vector should be identical to the vector y computed in Part ©. The only difference is that the indices of the values stored in y2 have changed. Determine this set of indices, and store them in the vector ny2. Plot y2[n] using stem(ny2,y2). Your plot should agree with figure 3.3.

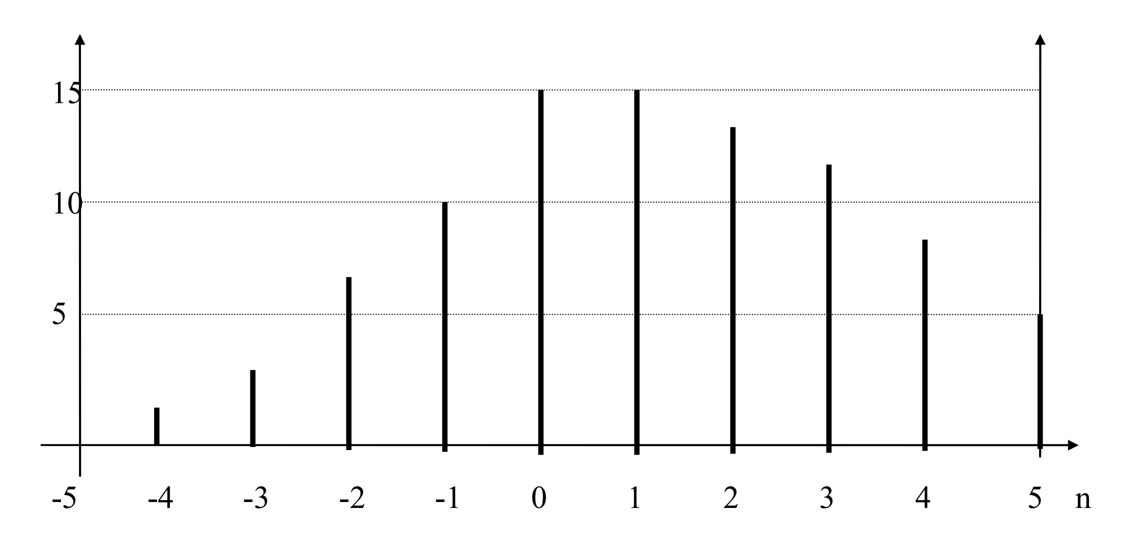


Fig: 3.3 Plot of the signal y[n]=x[n]*h[n+5]

Experiment # 4

Linear Time-Invariant Systems

Discrete-Time convolution

The convolution of discrete-time sequences h[n] and x[n] is represented mathematically by the expression

$$y[n] = \sum_{m=-\infty}^{\infty} h[m]x[n-m]$$

which can be viewed pictorially as the operation of flipping the time axis of the sequence h[m] and shifting it by n samples, then multiplying h[n-m] by x[m] and assuming the resulting product sequence over m. Since LTI system can be represented by its response to a single impulse, the output of an LTI system corresponds to the superposition of the responses of the system to each of the delayed and scaled impulses used to construct x[n].
In this program you will define some discrete-time signals and the impulse responses of some discrete-time LTI systems. Then the output of the LTI systems can be computed using *conv*.

a-

Since the MATLAB function *conv* does not keep track of the time indices of the sequences that are convolved, you will have to do some extra bookkeeping in order to determine the proper indices for the result of the *conv* function. For the sequences $h[n] = 2\delta[n+1] - 2\delta[n-1]$, and $x[n] = \delta[n] + 2\delta[n-2]$ construct vectors h and x. Define $y[n] = x[n] * h[n]$ and compute y = *conv*(h,x). Determine the proper time indexing for y and store this set of time indices in the vector ny. Plot y[n] as a function of n using *stem(ny,y).*

b-

Consider two finite-length sequences h[n] and x[n] that are represented in MATLAB using the vectors h and x, with corresponding time indices given by nh=[a:b] and nx=[c:d]. The function call y=*conv*(h,x) will return in the vector y the proper sequence values of y[n] = h[n] * x[n], however you must determine a corresponding set of time indices ny. To help you construct ny, consider the sequence $h[n] = \delta[n-a] + \delta[n-b]$ and $x[n] = \delta[n-c] + \delta[n-d]$. Determine analytically the convolution y[n] = h[n] * x[n]. From your answer, determine what ny should be in terms of a,b,c, and d. To check your result, verify that y[n] is of length M + N - 1 when a = 0, b = N - 1, c = 0, and d = M - 1.

c-

Consider an input x[n] and unit impulse response h[n] given by

$$x[n] = \left(\frac{1}{2}\right)^{n-2} u[n-2],$$
$$h[n] = u[n+2].$$

If you wish to compute y[n] = h[n] * x[n] using *conv*, you must deal appropriately with the infinite length of both x[n] and h[n]. Store in the vector x the values of x[n] for 0 <= n <= 24 and store in the vector h the values of h[n] for 0 <= n <= 14. Now store in the vector y the result of the function call *conv*(h,x). Since you have truncated both h[n] and x[n], argue that only a portion of the output of *conv* will be valid. Specify which values in the output are valid and which are not. Determine the values of the parameters a, b, c, and d such that nx = [a:b] and nh = [c:d], and use your answer from Part (b) to construct the proper time indices for y. Plot y[n] using *stem*, and indicate which values of y[n] are correct, and which value are invalid. Be sure to properly label the time axis for y[n].

Next consider what is known as block convolution which is often used in real-time implementation of digital filters such as speech or music processing systems where short processing delays are desired. This technique is most useful when processing a very long input sequence with a relatively short filter. The input sequence is broken into short blocks each can be processed independently with relatively little delay.
To illustrate the procedure, assume that you have a filter with a finite-length impulse response h[n] which is nonzero only on the interval 0 <= n <= P-1. Also assume that the input sequence x[n] is 0 for n < 0 and that the length of x[n] is significantly greater than P.

$$x[n] = \sum_{r=0}^{\infty} x_r[n - rL] \quad \textit{where} \;\; L > P,$$
$$\textit{and}$$
$$x_r = \begin{cases} x[n + rL], & 0 \le n \le L\text{-}1 \\ 0, & \textit{otherwise,} \end{cases}$$

d-

For $h[n] = (0.9)^n (u[n] - u[n-10])$ and $x[n] = \cos(n^2)\sin(2\pi n/5)$, compute y[n]=h[n]*x[n] for 0 <= n <= 99 directly using *conv*. Make a plot of y[n] over this range using *stem*.

e-

For L=50, break the sequence x[n] into two sequences, each of length 50. Compute y0[n]=h[n]*x0[n], and y1[n]=h[n]*x1[n], where x0[n] contains the first 50 samples of x[n] and x1[n] contains the second 50 samples of x[n]. The form of the output y[n] is given by

y[n]=x[n]*h[n]=y0[n]+y1[n-k].

Determine the appropriate value of k to use, and note that y0[n] and y1[n] will both be of length L+P-1. When y0[n] and y1[n] are added together, there will generally be a region where both are nonzero. It is for this reason that this method of block convolution is called the

overlap-add method. Compute y[n] in this manner, and plot y[n] over the range 0<=n<=99. Is your result the same as what you found in Part (d)?

Experiment # 5

Fourier Series

5.1 Objective

The objective of this laboratory is to gain a better heuristic understanding of the convergence of the Fourier series to a periodic function and to investigate how that signal is modified by passing it through a circuit.

5.2 Experiment Preparation

5.2.1 Basic Theoretical Principles

The mathematician J. B. Fourier showed that any “ properly-behaved” periodic function can be represented to any desired degree of accuracy by a sum of sinusoidal components. It can also be shown that in a “Lest-squares” sense, there is no better representation for a function than the Fourier series representation. The Fourier series representation for a function f(t) (in its sinusoidal form) is:

$$f(t) = a_0 + \sum_{k=1}^{\infty}\left[a_k \cos(k\omega_0 t) + b_k \sin(k\omega_0 t)\right]$$

where wo is the fundamental frequency of the function; that is,

$$\omega_0 = \frac{2\pi}{T}$$

Where T is the fundamental period of the function.

The Fourier coefficients, the a’s and b’s in the equation, may be computed by the following set of integrals:

$$a_0 = \frac{1}{T}\int_0^T f(t)dt$$

$$a_k = \frac{2}{T}\int_0^T f(t)\cos(k\omega_0 t)dt$$

$$b_k = \frac{2}{T}\int_0^T f(t)\sin(k\omega_0 t)dt$$

Now that we have a way to represent the function, we can use the Fourier series just as if it were the function itself and investigate its behavior in electronic systems. In particular, we can propagate the Fourier series representation of a signal through a

circuit's transfer function. Then, by using superposition, we can see what the output looks like by summing over all of the sinusoidal components.

For a single input frequency, we can obtain the steady-state output, which is the only one we are interested in for investigating the periodic behavior of a circuit, directly from knowing the input and the transfer function. If the input looks like

$$f(t) = A\cos(\omega t + \theta)$$

then by passing it through a transfer function H(s), we can obtain the steady-state output from knowing the magnitude and phase of H(s) evaluated as s = jw by

$$f_{ss}(t) = A|H(j\omega)|\cos(\omega t + \theta + \phi(j\omega))$$

For a Fourier-series representation of the input, we have

$$f(t) = a_0 + \sum_{k=1}^{\infty}\left[a_k \cos(k\omega_0 t) + b_k \sin(k\omega_0 t)\right]$$

as the input function and the steady-state output after propagation through the transfer function becomes

$$f(t) = a_0|H(j0)| + \sum_{k=1}^{\infty}\left[a_k|H(jk\omega_0)|\cos(k\omega_0 t + \phi(jk\omega_0)) + b_k|H(jk\omega_0)|\sin(k\omega_0 t + \phi(jk\omega_0))\right]$$

5.3 Experiment Procedure

1.The first part of this experiment of computing the Fourier-series representation for the function shown in figure 5.1. For simplicity, we will assume that the period of the function is 1. To compute the Fourier-series representation, you need first to compute the coefficients as a function of k and then write a program to compute the Fourier-series representation over one cycle and let the number of coefficients take on the values 1, 5, 10, 50, 100.

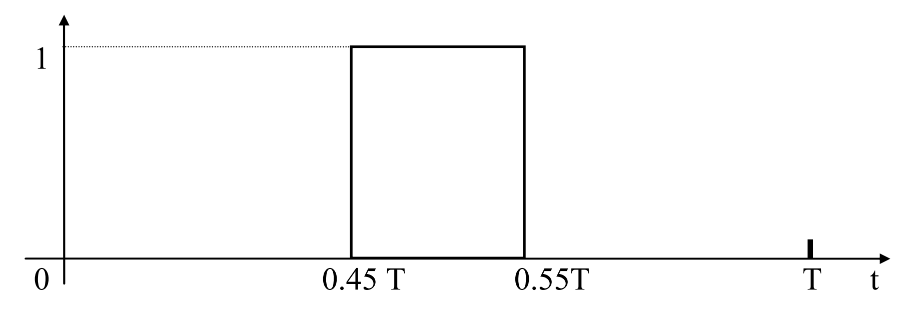


figure 5.1

2.Next, plot amplitude spectrum for the first 100 coefficients. This involves plotting the quantity

$$A_k = \sqrt{a_k^2 + b_k^2}$$

for k = 0, 100.

3.Next, we will look at how the amplitude spectrum becomes modified by the transfer function. The transfer function we will use is for a low-pass filter having a critical frequency a 10 rad/sec. This transfer function

$$H(s) = \frac{0.01586}{s^2 + 1.414s + 100}$$

Using this transfer function, plot the modified amplitude spectrum for the steady-state output. That is plot

$$A_k^{'} = \left|H(jk\omega_0)\right| A_k$$

for k = 0, 100.

4.Finally, plot the full steady-state output for the input modified by the transfer function for the same values of k used in the first part of this experiment.

5.4 Experiment Evaluation

1.Notice the mis-match between the fit and the actual function around the corners (this is called the Gibb's phenomenon). Does it go away as the number of components increases?

2.Based upon the amplitude spectrums before and after filtering, is there a point in the series summation after which you would say that adding more components does not affect the outcome? How might you determine that points?

Experiment # 6

Fourier Transform

6.1 Objective

The purpose of this laboratory is to observe the effect of sampling on the Fourier transform of a function. This will be done via a window function.

6.2 Experiment Preparation

6.2.1 Basic Theoretical Principles

Fourier Transforms and Window Function

The Fourier transform allows us to represent a non-periodic function in the frequency domain. The usual definition of Fourier transform. F(w), of a function f(t) is

$$F(\omega) = \int_{-\infty}^{\infty} f(t) e^{-j\omega t} dt$$

Due to this definition, F(w) and f(t) are often referred to as Fourier transform pairs. To aid us in our quest for the computation of the transform, we will re-write the defining integral as

$$F(\omega) = \int_{-\infty}^{\infty} f(t)\cos(\omega t)dt \ - j\int_{-\infty}^{\infty} f(t)\sin(\omega t)dt$$

which is at least in terms of functions which the computer knows how to evaluate. You should notice that the Fourier transform is, in general, a complex quantity. We can then talk in terms of the magnitude and phase of the transform as a function of frequency using the usual definitions of magnitude and phase for complex numbers. As you have noticed, the defining integral in over all t which in this case would correspond to the life of the Universe and then some. Needless to say, we cannot sample a real function for such a long time so we need to make some approximations. One of these approximations is that if we sample f(t) for a long but finite time them we will obtain a reasonable approximation to the infinite integral. The only question at this point is how long is long enough. The only good answer is by trial and error. The way we will perform this trial and error process is by “windowing the f(t) and allowing the window width to grow. A window function looks like.

w(a,b) = u(t-a) - u(t-b)

where the u( ) are the unit step functions. For our case here, we will let a = - T/2 and b = T/2 where T is the window width. The computation of the Fourier transform will now look like the following with the window added:

$$F(j\omega) = \int_{-\infty}^{\infty} f(t)\omega(a,b)\cos(\omega t)dt \ - \ j\int_{-\infty}^{\infty} f(t)\omega(a,b)\sin(\omega t)dt$$

or

$$F(j\omega) = \int_{a}^{b} f(t)\cos(\omega t)dt \ - \ j\int_{a}^{b} f(t)\sin(\omega t)dt$$

The price we must pay for this integration is effect this process has on the resulting Fourier transform. The multiplication of f(t) by a window function has the effect of convolving the transform of the window function, W(w), with the transform of f(t), F(w). The Fourier transform of the window function will look like a sin(x)/x or sinc(x) function.

6.3 Experiment Procedure

This experiment is to investigate the windowing effect on the computation of the Fourier transform for a simple function, namely,

f(t) = 1 sin(100t)

In this experiment, we will let the window width T take the values of 2pi/100, 20pi/100, 200pi/100, and 2,000pi/100. That is, we will be computing the transform based on the sampling of 1, 10, 100, and 1,000 cycles of the input frequency. You are to compute and plot (magnitude and phase) the Fourier transform of f(t) with the given window values. Use your knowledge of the exact Fourier transform of a sinusoidal to set your plotting parameters.

6.4 Experiment Evaluation

1. How did the width of the window transform change in frequency space as the width of the window was changed in the time-domain?
2. What would you expect the sampled transform to look like if there were several frequencies present?

Experiment # 7

Laplace Transform

7.1 Objective

The purpose of this lab to gain familiarity with Laplace transforms, including the Laplace transforms of step functions and related functions.

7.2 Experiment Preparation

7.2.1 Basic Theoretical Principles

Fourier Transforms and Window Function

The Laplace transform is the tool that makes it possible to represent an arbitrary input X(t) in terms of exponential components (e^{st}).

The Laplace transform can be used to solve differential equations of any order and also to find the output of electrical circuits.

For a signal x(t):

$$X(S)= L(x(t))= \int x(t)\, e^{-st}\, dt$$

Where, X(s) is the Laplace transform of x(t), And s is the complex frequency.

And

$$x(t)= 1/(2\, pi\, {*}j)\ \int X(s)\, e^{st}\, ds$$

x(t) is the inverse Laplace transform of X(s).

$$x(t) \longleftrightarrow X(s)$$

$$s=\sigma+j\omega$$

Note that

when $\sigma=0$; $S=j\omega$, this leads to Fourier Transform.

Note: the Fourier Transform is a special case of the Laplace Transform.

7.3 Experiment Procedure

In this lab, we will

- Find the Laplace transform of these functions:

1. x(t)= u(t).
2. x(t)= t u(t).
3. x(t)= sin t , $-\pi < t < \pi$

- Use the residue command to find the inverse Laplace transform of the following functions:

1. $X1(s)= (2s^2+5)/(s^2+3 s+2)$.

```
Matlab code:
Num=[2 0 5];
Den=[1 3 2];
[r, p, k ]=residue(Num,Den)
```

Repeat for the following:

2. $X2(s)= (2s^2+7s+4)/(s+1)(s+2)^2$
3. $X3(s)= (8 s^2+21s+19)/(s+2)(s^2 +s+7)$.

7.4 Experiment Evaluation

Experiment # 8

Frequency Response and Filters

8.1 Objective

The objective of this laboratory is to look at the frequency responses of any circuit from the coefficients of the system transfer function polynomial.

8.2 Experiment Preparation

8.2.1 Basic Theoretical Principles

The frequency response of a system is the major way of characterizing how a system behaves in the frequency domain. It is important to understand the frequency characteristics of a given system rather than time domain characteristics alone for many practical applications like filter design.

The frequency response of a system, $H(\omega)$ is just the transfer function, $H(s)$ evaluated at $s \to j\omega$. Frequency response is usually a complex valued function, so it can be written as $H(\omega) = |H(\omega)|\angle H(\omega)$, where $|H(\omega)|$ is the magnitude response and $\angle H(\omega)$ is the phase response.

RLC Circuit Transfer Functions

For any system, the transfer function is defined to be the ratio of the output quantity of interest to the input driver. Typically, the transfer function will be the ratio of the output voltage across a component like a resistor or capacitor to the input driving voltage or current source. The units on the transfer function will then correspond to a gain (unitless) or a resistance or impedance depending upon the input/output combination. The circuits shown on figures 8.1, 8.2, and 8.3 are RLC filters for a high-pass, low-pass, and band-pass filter. We have derived for you the transfer function for each circuit in terms of RLC component values. The equations have taken the typical s-domain results and evaluated the transfer functions at s = jw. That is, we are looking at the pure frequency response of the transfer function.

In the s-plane, this corresponds to traveling up and down the imaginary axis. Also given is the transfer function in terms of the ratio of second-order polynomials. We will see how to use this form below.

Circuit 1

$$\frac{V_0}{V_{in}} = \frac{R}{R + R_L + j\omega L + \dfrac{1}{jC\omega}} = \frac{jRC\omega}{LC(j\omega)^2 + C(R+R_L)j\omega + 1}$$

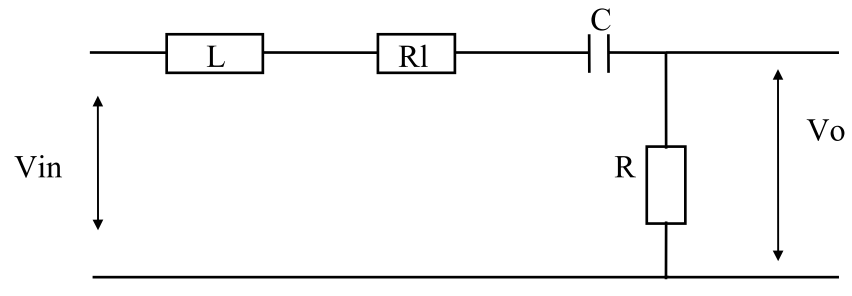


Figure 8.1

Circuit 2

$$\frac{V_0}{Vin} = \frac{R + \frac{1}{jC\omega}}{R + R_l + jL\omega + \frac{1}{jC\omega}} = \frac{1 + RC(j\omega)^2}{1 + (R + R_l)jC\omega + LC(j\omega)^2}$$

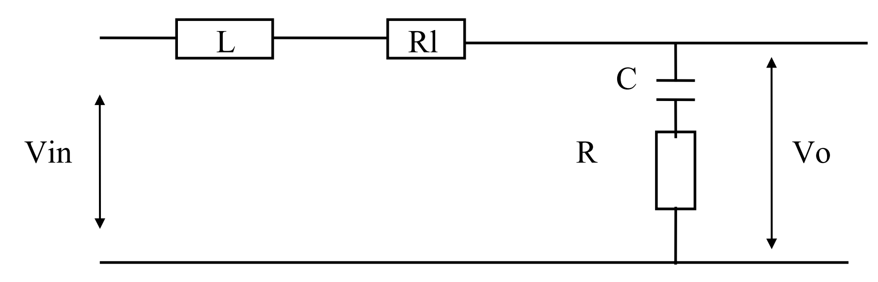


Figure 8.2

Circuit 3

$$\frac{V_0}{Vin} = \frac{R + R_l + jL\omega}{R + R_l + jL\omega + \frac{1}{jC\omega}} = \frac{C(R + R_l) + j\omega + LC(j\omega)^2}{1 + C(R + R_l)j\omega + LC(j\omega)^2}$$

C

R

Vin

Rl

Vo

L

Figure 8.3

In this experiment, we will be looking at the frequency response of the circuits and plotting the response as two components: the amplitude and the phase responses. The amplitude plot will be a plot of 20 times the base-10 logarithm of the magnitude of the frequency response evaluated at s = jw (20 log|H(jw)|). The phase plot will be the phase angle of the frequency response of the transfer function evaluated at s = jw.

8.3 Experiment Procedure

In this lab you are asked to find the frequency response(magnitude and phase) for the three circuits.

Let R1=1KΩ
R= 1KΩ
C=1μF
L=2 mH

For the first circuit the Matlab code is:

```
Num=[1,0];
Den=[2*10^-6, 2, 1];
W=[0:.1:2*pi];
freqs(Num,Den,W)
% you can use the command bode
% bode(Num,Den)
```

8.4 Experiment Evaluation

When the plots are complete, record the following information from your graphs:

1. The cutoff frequencies, roll-off rate for the low-pass, band-pass, and high-pass filters (Need to show cutoff frequencies, roll-off rate on your plots).
2. The quality factor Q, the lower and upper cutoff frequencies for the band-pass filter (Show the locations of cutoff frequencies on your plots).
3. The filter gain (difference in dB units between the maximum of the amplitude curve and 0 dB) for all the three filters.

Experiment # 9
Frequency Response and Convolution
(Simulink)

Objective: In this lab, we will explore the frequency response of a continuous-time system with Simulink. As studied in lecture

The frequency response of a continuous-time system is given by

$$H(j\Omega) = \frac{Y(j\Omega)}{X(j\Omega)}$$

The frequency response of a continuous-time system can be measured in lab by connecting a sine wave generator to the input of system and comparing the input with the output. If x(t) is the input, a sine wave of frequency Ω, and y(t) is the output, since the system is linear and time invariant, y(t) will also be a sinusoid having the same frequency as x(t). Moreover, y(t) will differ from x(t) by a scale factor and a phase shift. The scale factor is

$$|H(j\Omega)|$$

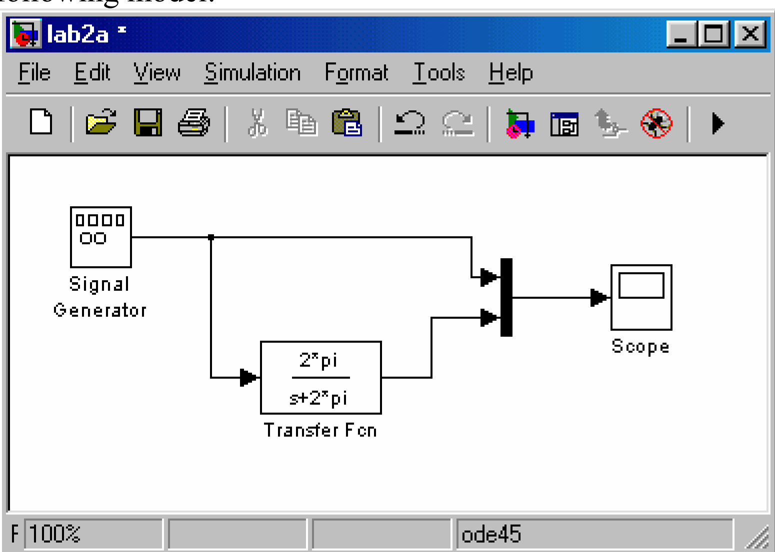
and the phase shift is

$$\arg(H(j\Omega)).$$

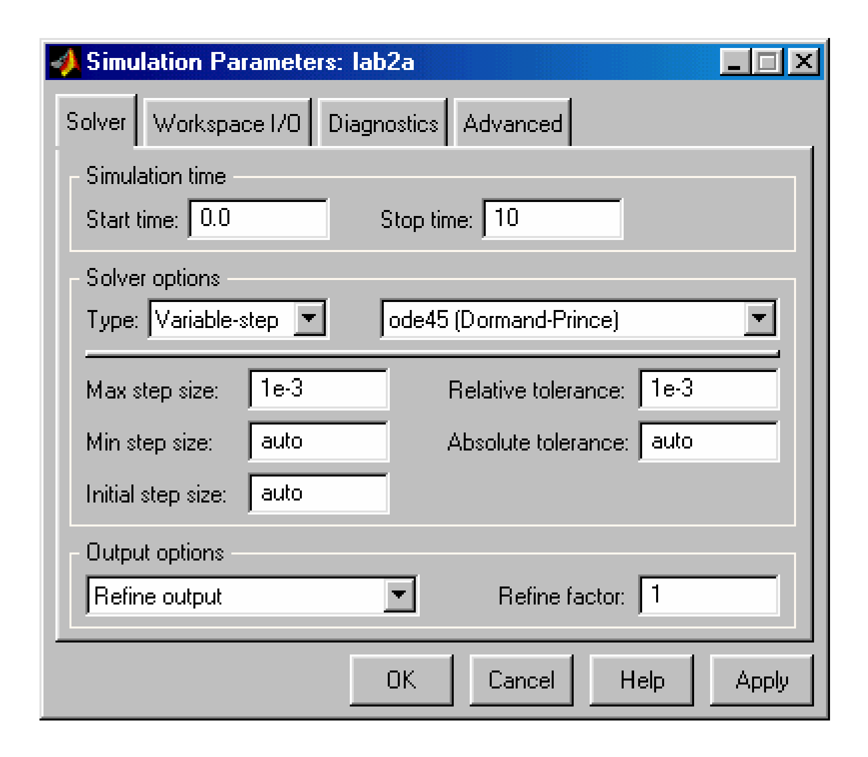
In other words

$$Y(t)=|H(j\Omega)| \cos(\Omega t+\arg(H(j\Omega)))$$

Put together the following model:



The MUX allows you to view both the input and output of the filter on the scope. The input will appear in yellow, the output in purple. Set the simulation parameters to:

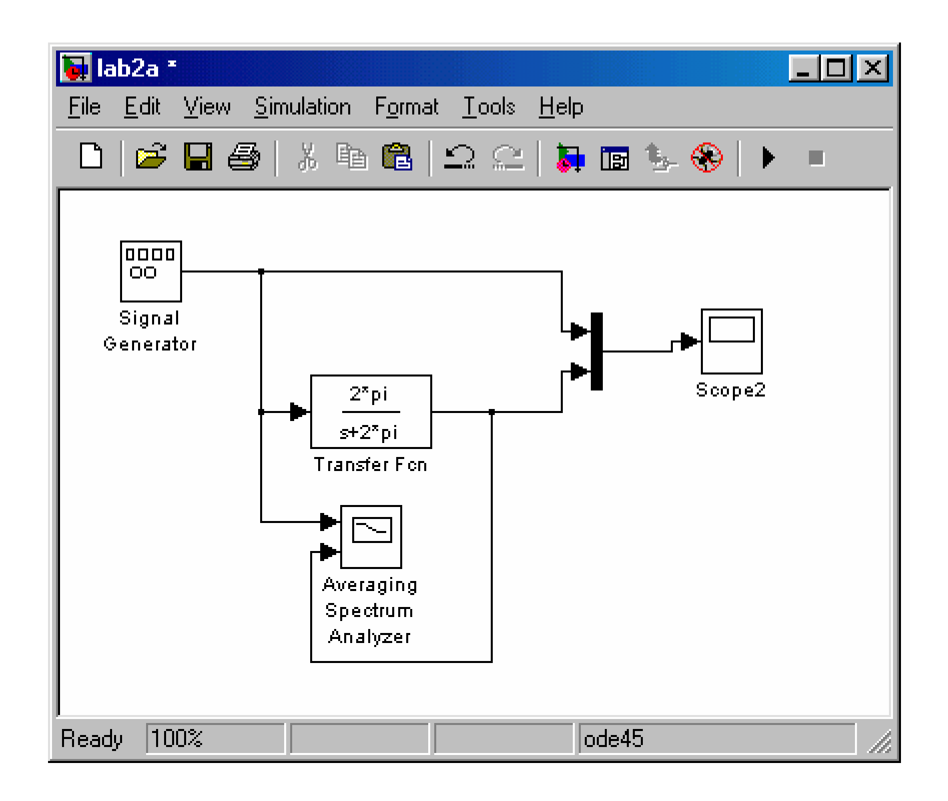


And run a series of experiments which will enable you to measure and plot the magnitude and phase (in radians) of the frequency response of the given transfer function (i.e. filter). Make sure you measure the magnitude and phase over a sufficiently large number of frequencies to get a smooth plot (the range of frequencies should be from 1 to 30 rad/sec in increments of 2 rad/sec). The phase can be obtained from the time difference between successive peaks of the input and output of the filter and knowledge of the frequency. The phase shift can be easily determined from the time shift as follows:

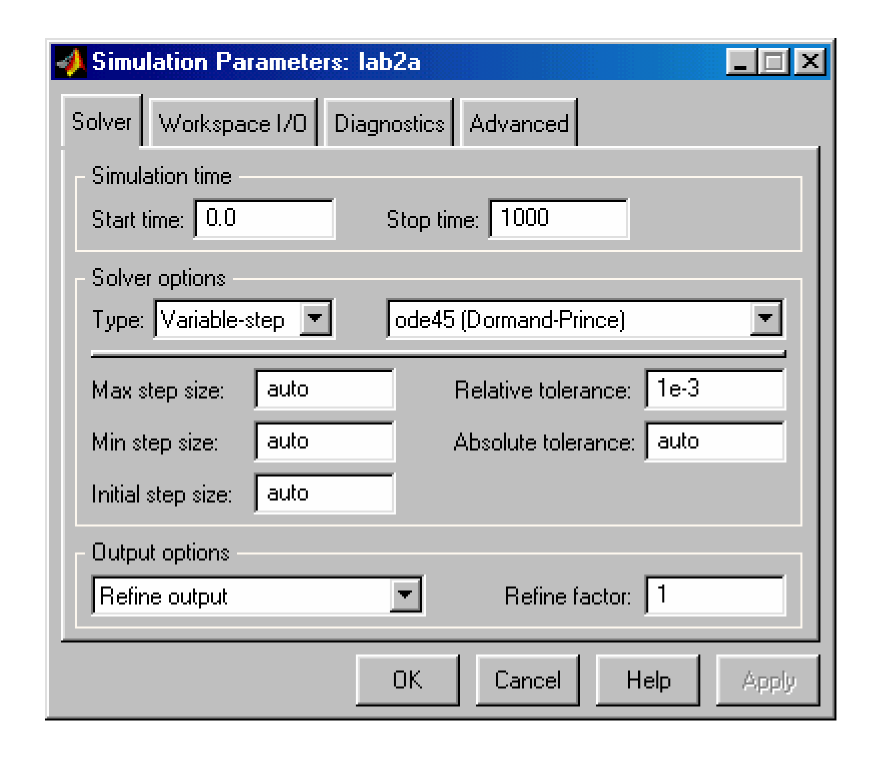
$$\begin{aligned} y(t) &= Kx(t-\Delta t) \\ &= K\cos\left(\Omega_0\left(t-\Delta t\right)\right) \\ &= K\cos\left(\Omega_0 t-\Omega_0 \Delta t\right) \end{aligned}$$

Comparing (2) and (1), we find that magnitude of the frequency response is $\left|H\left(j\Omega\right)\right| = K$ and the phase of the frequency response is
$\angle H\left(j\Omega\right) = -\Omega_0 \Delta t$. You can use an Excel spreadsheet or Matlab to plot your results. The simulation stop time can be set so that you see about 5 cycles of x(t) and y(t). Also ignore the initial transients in y(t), the amplitude should correspond to its steady-state value. What type of filter is this?

Next go to "Simulink Extras" in the Library Browser and attach the "Averaging Spectral Analyzer" as shown:



Change the signal in the signal generator block from the Sine wave to the random signal. The random signal is also called "white noise" which contains all frequencies in equal amounts. The averaging spectrum analyzer will therefore effectively perform the same measurements you just performed by hand. The Averaging spectrum analyzer finds the Fourier transform of the output: Y(jΩ), averaged over many data segments. It will then find the average Fourier transform of the input, X(jΩ), and will then plot the magnitude and phase of the ratio, H(jΩ) = Y(jΩ)/X(jΩ) Run the simulation for about 1000 seconds, i.e. set the simulation parameters to:

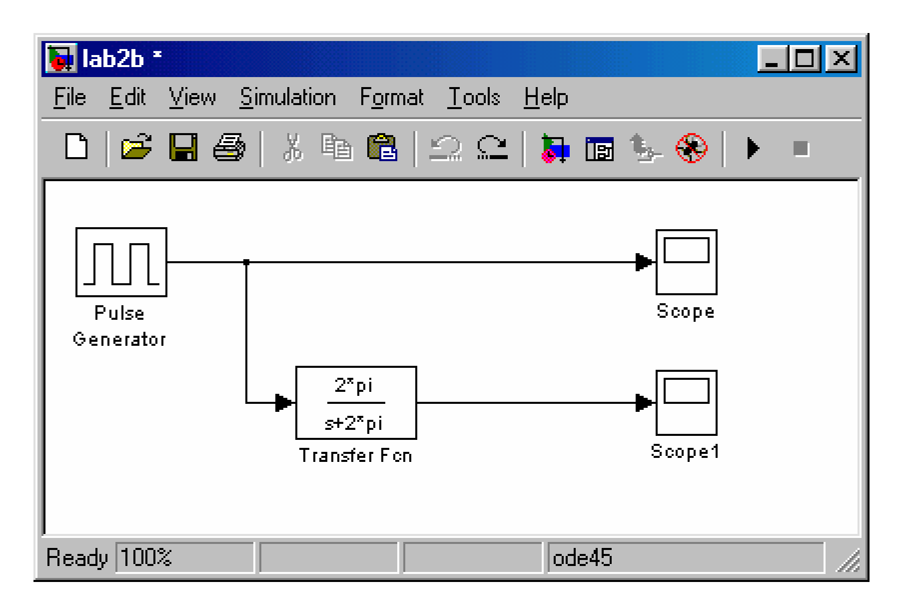


and copy the resulting plot into your write-up.

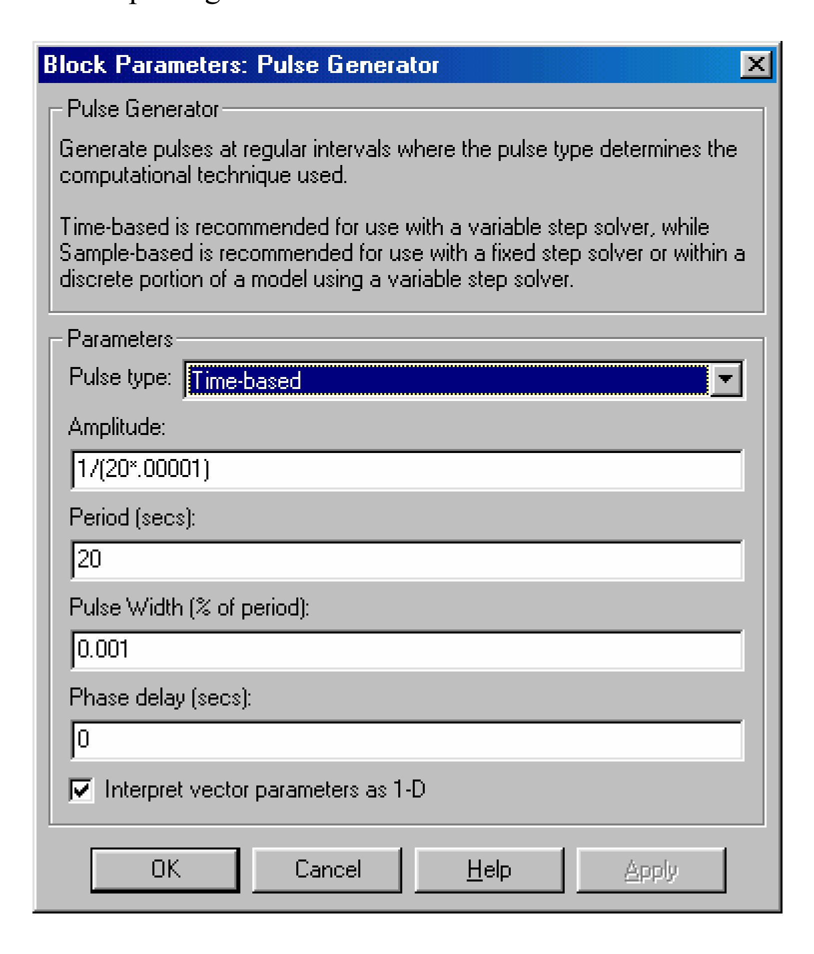
Next find the theoretical transfer function (set $s = j\Omega$) and use Matlab to plot its magnitude and phase. Include these theoretical plots in your write-up and comment on any differences between your theoretical and experimental results. A program for doing this plot is:

```
N = 100;
for k = 1:N,
    Omega(k) = k/N*30;
    H(k) = 2*pi/(2*pi + j*Omega(k));
end
figure(2)
plot(Omega,abs(H))
figure(3)
plot(Omega,angle(H))
```

Next we'll examine the input-output behavior in the time domain for more general types of signals. Connect the following model:



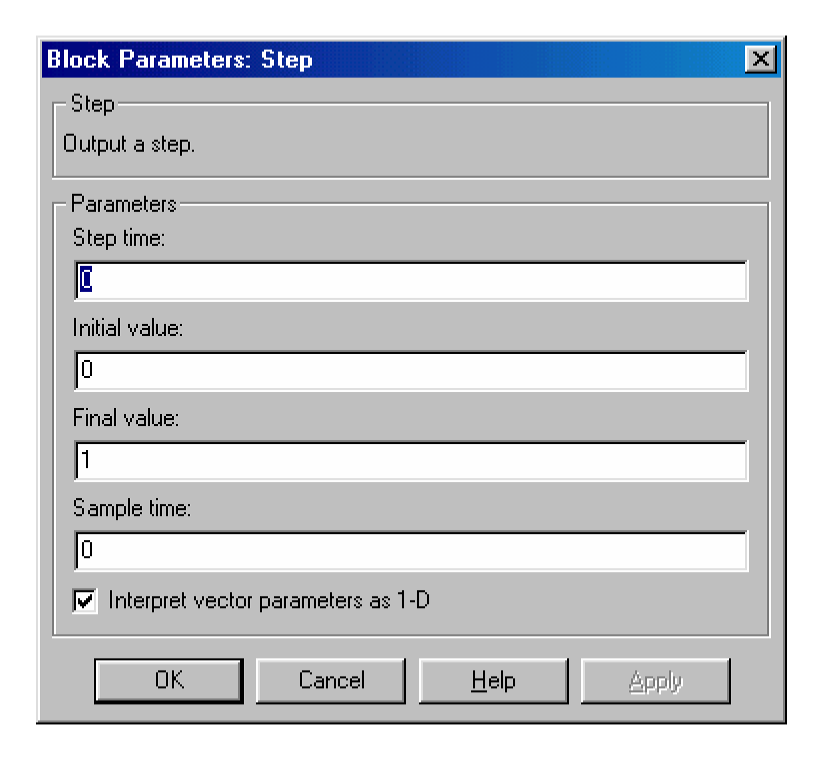
Set the parameters of the pulse generator to:



The pulse generator is set up to mimic an impulse function. What is the area under the impulse? Run the simulation for two seconds and plot the input and the impulse response of the filter. How does it compare with theoretical impulse response? Next replace the pulse generator with a step function generator. Recall the definition of the step function, u(t) from lecture:

$$u(t) = \begin{cases} 1, & t \geq 0 \\ 0, & t < 0 \end{cases}$$

Set the parameters for the step function (u(t)) to:



Run the simulation for 2 seconds and plot the input and output of the filter. Does the result agree with the theoretically predicted output based on the convolution

Experiment # 10
Fourier Series (Simulink)

Objective: To understand the functions of Simulink and demonstrate the properties of the Fourier series.

A sine wave and a pulse generator were connected to a scope so that they could be viewed.

How many cycles of the square wave appear in the scope window? What is its period?

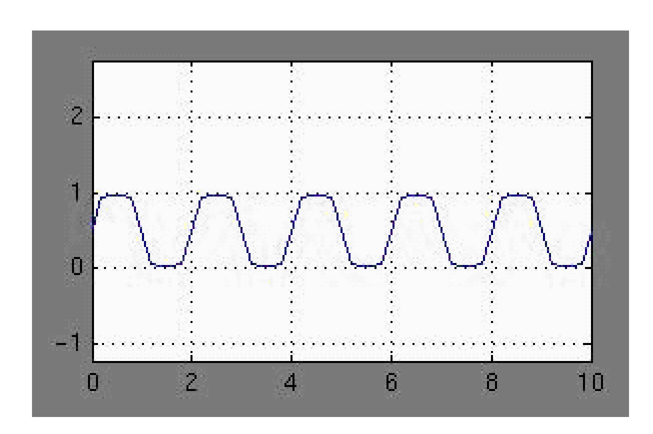
The pulse generator was connected to several sine wave generators and integrators to simulate the coefficients for the Fourier series of the square wave produced by the Pulse Generator. The values for the coefficients are listed below.

b1 = 0.6366
b2 = 5.856e-11
b3 = 0.2122
b4 = 1.918e-09
b5 = 0.1273
b6 = 1.511e-08
b7 = 0.09095
b8 = 6.685e-08
b9 = 0.07074
b10 = 2.161e-07

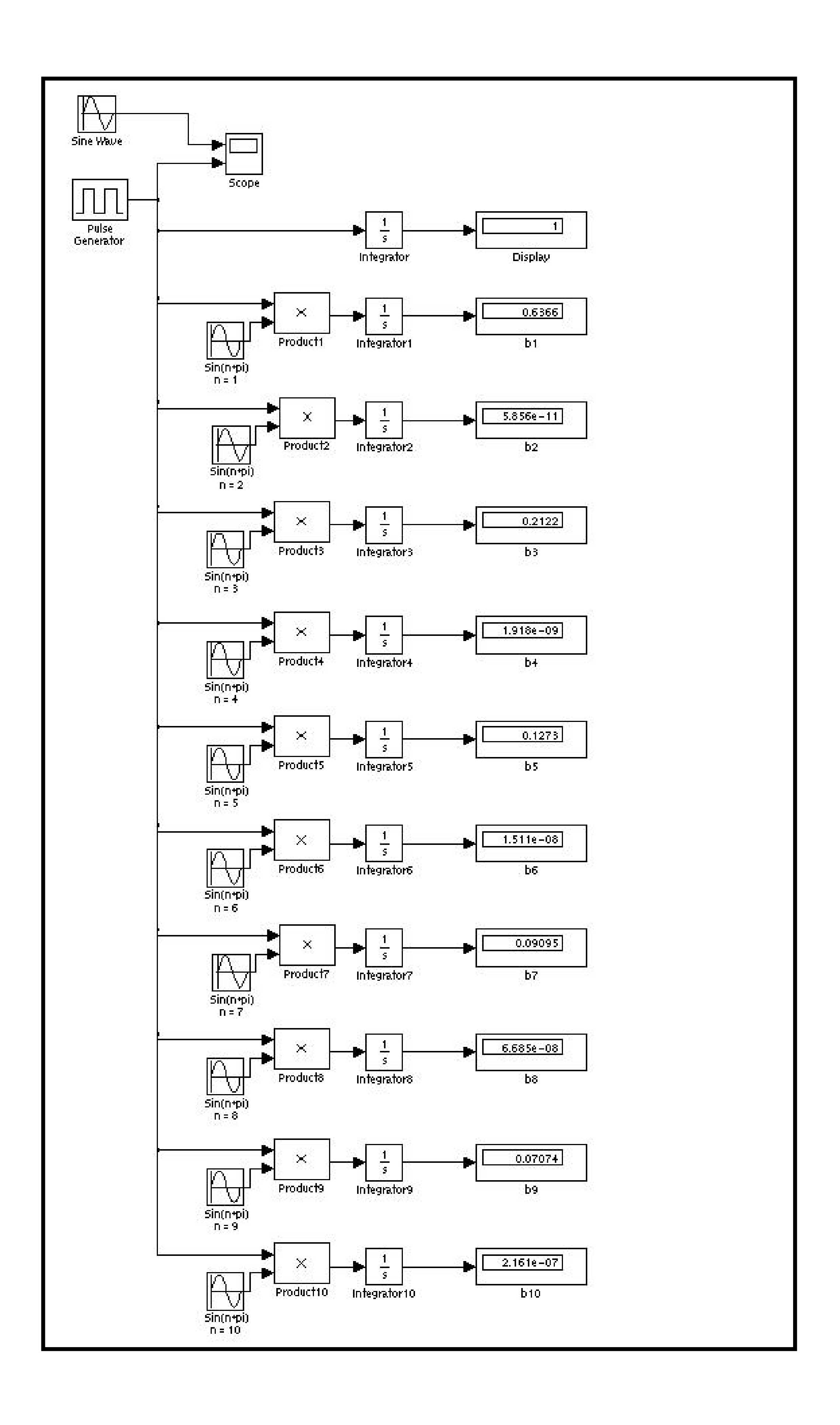
The coefficients were combined with sine waves and added together to try and reproduce the square wave according to the following equation:

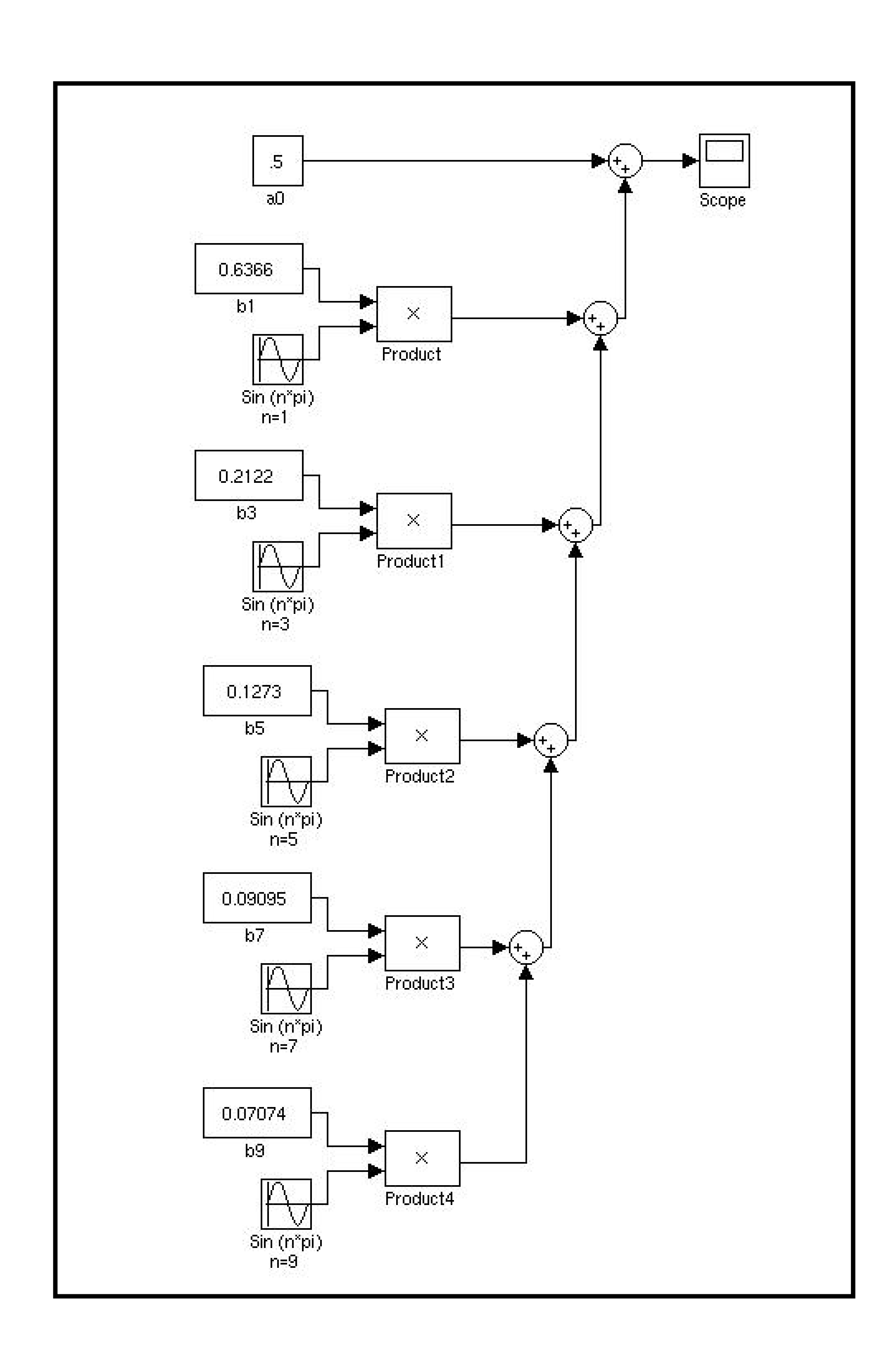
x(t) = a0 + b1*Sin(pi) + b3*Sin(3*pi) + b5*Sin(5*pi) + ...

The generated wave is shown below.



Generated Square wave


Sine Wave
Scope
Pulse Generator
Integrator
1
Display
Product1
Integrator1
0.6366
b1
Sin(n*pi) n = 1
Product2
Integrator2
5.856e-11
b2
Sin(n*pi) n = 2
Product3
Integrator3
0.2122
b3
Sin(n*pi) n = 3
Product4
Integrator4
1.918e-09
b4
Sin(n*pi) n = 4
Product5
Integrator5
0.1273
b5
Sin(n*pi) n = 5
Product6
Integrator6
1.511e-08
b6
Sin(n*pi) n = 6
Product7
Integrator7
0.09095
b7
Sin(n*pi) n = 7
Product8
Integrator8
6.685e-08
b8
Sin(n*pi) n = 8
Product9
Integrator9
0.07074
b9
Sin(n*pi) n = 9
Product10
Integrator10
2.161e-07
b10
Sin(n*pi) n = 10


.5
a0
Scope
0.6366
b1
Product
Sin (n*pi)
n=1
0.2122
b3
Product1
Sin (n*pi)
n=3
0.1273
b5
Product2
Sin (n*pi)
n=5
0.09095
b7
Product3
Sin (n*pi)
n=7
0.07074
b9
Product4
Sin (n*pi)
n=9